

## Лабораторная работа №6

### Темы:

1. Изучение аффинных преобразований в пространстве.
2. Изучение принципов построения 3D – изображений.
3. Изучение метода удаления невидимых граней с помощью их сортировки по глубине (алгоритм художника).

**Цель:** работа с аффинными преобразованиями, реализация 3D графиков.

### Задание:

Реализовать класс **class CPlot3D** для изображения поверхности, которая может быть описана однозначной функцией двух переменных

### Ход работы:

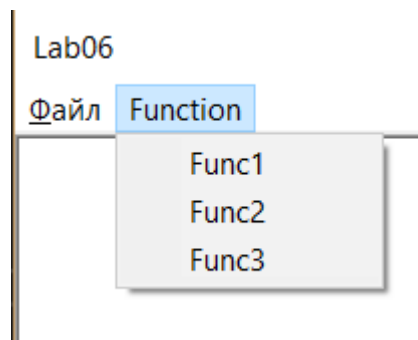
Изучить теоретическую часть лабораторной работы. В конце документа литература.

Создать приложение Windows для изображения поверхностей:

1.  $f_1(x, y) = x^2 + y^2$ ,  $x \in [-5; 5]$ ;  $y \in [-5; 5]$ ;  $\Delta x = 0,25$ ;  $\Delta y = 0,25$
2.  $f_2(x, y) = x^2 - y^2$ ,  $x \in [-5; 5]$ ;  $y \in [-5; 5]$ ;  $\Delta x = 0,25$ ;  $\Delta y = 0,25$
3.  $f_3(x, y) = \sqrt{1 - x^2 - y^2}$

Значения функции  $f_3(x, y)$  рассчитываются в сферической системе координат.

Каждая из поверхностей должна отображаться при выборе соответствующего пункта меню:



Для выполнения первого задания данной лабораторной работы необходимо создать класс CPlot3D в файле LibSurface.

LAB06 ChildView - создание окна

stdafx.h - подключаем нужные библиотеки, компоненты MFC

LibSurface - создание главного окна

LibGraph- на всех плоскостях координаты вычисляются

ChildView - функции рисования реализации

Для начала нам нужно создать и реализовать функционал создания окна приложения и меню выбора нужных функций. Для этого нужны заголовочные файлы: Lab06.h, MainFrm.h, LibGraph.h и LibSurface.h. Реализация соответственно в файлах .cpp.

Создаем главный заголовочный файл stdafx.h, в котором содержатся все нужные библиотеки и файлы расширения .h :

```
// stdafx.h: включите файл для добавления стандартных системных файлов
//или конкретных файлов проектов, часто используемых,
// но редко изменяемых

#pragma once

#ifndef VC_EXTRALEAN
#define VC_EXTRALEAN           // Исключите редко используемые компоненты из
заголовков Windows
#endif

#include "targetver.h"

#define _ATL_CSTRING_EXPLICIT_CONSTRUCTORS      // некоторые конструкторы CString
будут явными

// отключает функцию скрытия некоторых общих и часто пропускаемых предупреждений MFC
#define _AFX_ALL_WARNINGS

#include <afxwin.h>           // основные и стандартные компоненты MFC
```

```
#include <afxext.h>           // расширения MFC

#include <FLOAT.H> // Для DBL_MAX , DBL_MIN
#include <fstream>
#include <math.h>
#include "CMatrix.h"
#include <vector>

#include "LibGraph.h"
#include "LibSurface.h"
```

```

#ifndef _AFX_NO_OLE_SUPPORT
#include <afxdtctl.h>           // поддержка MFC для типовых элементов управления
Internet Explorer 4
#endif
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>           // поддержка MFC для типовых элементов управления
Windows
#endif // _AFX_NO_AFXCMN_SUPPORT

#include <afxcontrolbars.h>     // поддержка MFC для лент и панелей управления

#ifdef _UNICODE
#if defined _M_IX86
#pragma comment(linker, "/manifestdependency:\"type='win32'
name='Microsoft.Windows.Common-Controls' version='6.0.0.0' processorArchitecture='x86'
publicKeyToken='6595b64144ccf1df' language='*'\")
#elif defined _M_X64
#pragma comment(linker, "/manifestdependency:\"type='win32'
name='Microsoft.Windows.Common-Controls' version='6.0.0.0'
processorArchitecture='amd64' publicKeyToken='6595b64144ccf1df' language='*'\")
#else
#pragma comment(linker, "/manifestdependency:\"type='win32'
name='Microsoft.Windows.Common-Controls' version='6.0.0.0' processorArchitecture='*
publicKeyToken='6595b64144ccf1df' language='*'\")
#endif
#endif

```

В файле Resource.h содержатся основные константы :

```

#define IDD_ABOUTBOX                100
#define IDP_OLE_INIT_FAILED        100
#define IDR_MAINFRAME              128
#define IDR_Lab01TYPE              130
#define ID_CPlot3D_FUNC1          32771
#define ID_CPlot3D_FUNC2          32772
#define ID_CPlot3D_FUNC3          32773

// Следующие стандартные значения для новых объектов
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE    312
#define _APS_NEXT_COMMAND_VALUE    32776
#define _APS_NEXT_CONTROL_VALUE    1000
#define _APS_NEXT_SYMED_VALUE      310
#endif
#endif

```

Lab06.h :

```

// Lab03.h: основной файл заголовка для приложения Lab03
//
#pragma once

#ifndef __AFXWIN_H__
    #error "включить stdafx.h до включения этого файла в PCH"
#endif

#include "resource.h"           // основные символы

```

```

// CLab01App:
// Сведения о реализации этого класса: Lab01.cpp
//

class CLab01App : public CWinApp
{
public:
    CLab01App() noexcept;

    // Переопределение
public:
    virtual BOOL InitInstance();
    virtual int ExitInstance();

    // Реализация

public:
    afx_msg void OnAppAbout();
    DECLARE_MESSAGE_MAP()
};

extern CLab01App theApp;

```

### Lab06.cpp:

```

// Lab02.cpp: определяет поведение классов для приложения.
//

#include "stdafx.h"
#include "afxwinappex.h"
#include "afxdialogex.h"
#include "Lab06.h"
#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CLab01App

BEGIN_MESSAGE_MAP(CLab01App, CWinApp)
END_MESSAGE_MAP()

// Создание CLab01App

CLab01App::CLab01App() noexcept
{
    SetAppID(_T("Lab01.AppID.NoVersion"));
}

// Единственный объект CLab01App

CLab01App theApp;

// Инициализация CLab01App

BOOL CLab01App::InitInstance()

```

```

{
    // InitCommonControlsEx() требуются для Windows XP, если манифест
    // приложения использует ComCtl32.dll версии 6 или более поздней версии для
    включения
    // стилей отображения. В противном случае будет возникать сбой при создании
    любого окна.
    INITCOMMONCONTROLSEX InitCtrls;
    InitCtrls.dwSize = sizeof(InitCtrls);
    // Выберите этот параметр для включения всех общих классов управления, которые
    необходимо использовать
    // в вашем приложении.
    InitCtrls.dwICC = ICC_WIN95_CLASSES;
    InitCommonControlsEx(&InitCtrls);

    CWinApp::InitInstance();

    // Инициализация библиотек OLE
    if (!AfxOleInit())
    {
        AfxMessageBox(IDP_OLE_INIT_FAILED);
        return FALSE;
    }

    AfxEnableControlContainer();

    EnableTaskbarInteraction(FALSE);

    // Для использования элемента управления RichEdit требуется метод
    AfxInitRichEdit2()
    // AfxInitRichEdit2();

    // Стандартная инициализация
    // Если эти возможности не используются и необходимо уменьшить размер
    // конечного исполняемого файла, необходимо удалить из следующего
    // конкретные процедуры инициализации, которые не требуются
    // Измените раздел реестра, в котором хранятся параметры
    // TODO: следует изменить эту строку на что-нибудь подходящее,
    // например на название организации
    SetRegistryKey(_T("Локальные приложения, созданные с помощью мастера
приложений"));

    // Чтобы создать главное окно, этот код создает новый объект окна
    // рамки, а затем задает его как объект основного окна приложения
    CFrameWnd* pFrame = new CMainFrame;
    if (!pFrame)
        return FALSE;
    m_pMainWnd = pFrame;
    // создайте и загрузите рамку с его ресурсами
    pFrame->LoadFrame(IDR_MAINFRAME,
        WS_OVERLAPPEDWINDOW | FWS_ADDTOTITLE, nullptr,
        nullptr);

    // Разрешить использование расширенных символов в горячих клавишах меню
    CMFCToolBar::m_bExtCharTranslation = TRUE;
}

```

```

        // Одно и только одно окно было инициализировано, поэтому отобразите и обновите
его
        pFrame->ShowWindow(SW_SHOW);
        pFrame->UpdateWindow();
        return TRUE;
    }

int CLab01App::ExitInstance()
{
    //TODO: обработайте дополнительные ресурсы, которые могли быть добавлены
    AfxOleTerm(FALSE);

    return CWinApp::ExitInstance();
}

// Обработчики сообщений CLab01App

// Диалоговое окно CAboutDlg используется для описания сведений о приложении

class CAboutDlg : public CDialogEx
{
public:
    CAboutDlg() noexcept;

    // Данные диалогового окна
#ifdef AFX_DESIGN_TIME
    enum { IDD = IDD_ABOUTBOX };
#endif

protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // поддержка DDX/DDV

    // Реализация
protected:
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() noexcept : CDialogEx(IDD_ABOUTBOX)
{
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialogEx::DoDataExchange(pDX);
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialogEx)
END_MESSAGE_MAP()

// Команда приложения для запуска диалога
void CLab01App::OnAppAbout()
{
    CAboutDlg aboutDlg;
    aboutDlg.DoModal();
}

// Обработчики сообщений CLab01App

```

## MainFrm.h :

```
// MainFrm.h: интерфейс класса CMainFrame
//

#pragma once
#include "ChildView.h"

class CMainFrame : public CFrameWnd
{
public:
    CMainFrame() noexcept;
protected:
    DECLARE_DYNAMIC(CMainFrame)

// Атрибуты
public:

// Операции
public:

// Переопределение
public:
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    virtual BOOL OnCmdMsg(UINT nID, int nCode, void* pExtra, AFX_CMDHANDLERINFO*
pHandlerInfo);

// Реализация
public:
    virtual ~CMainFrame();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected: // встроенные члены панели элементов управления
    CStatusBar      m_wndStatusBar;
    CChildView      m_wndView;

// Созданные функции схемы сообщений
protected:
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    afx_msg void OnSetFocus(CWnd *pOldWnd);
    DECLARE_MESSAGE_MAP()
};
```

## MainFrm.cpp:

```
// MainFrm.cpp: реализация класса CMainFrame
//

#include "stdafx.h"
#include "Lab06.h"

#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif
```

```

// CMainFrame

IMPLEMENT_DYNAMIC(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
    ON_WM_CREATE()
    ON_WM_SETFOCUS()
END_MESSAGE_MAP()

static UINT indicators[] =
{
    ID_SEPARATOR,          // индикатор строки состояния
    ID_INDICATOR_CAPS,
    ID_INDICATOR_NUM,
    ID_INDICATOR_SCRL,
};

// Создание или уничтожение CMainFrame

CMainFrame::CMainFrame() noexcept
{
    // TODO: добавьте код инициализации члена
}

CMainFrame::~CMainFrame()
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    //-----
    int width = 900, height = 680;
    MoveWindow((GetSystemMetrics(SM_CXSCREEN) / 2 - width / 2),
        (GetSystemMetrics(SM_CYSCREEN) / 2 - height / 2), width, height);
    //-----

    if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
        return -1;

    // создать представление для размещения рабочей области рамки
    if (!m_wndView.Create(nullptr, nullptr, AFX_WS_DEFAULT_VIEW, CRect(0, 0, 0, 0),
this, AFX_IDW_PANE_FIRST, nullptr))
    {
        TRACE0("Не удалось создать окно представлений\n");
        return -1;
    }

    if (!m_wndStatusBar.Create(this))
    {
        TRACE0("Не удалось создать строку состояния\n");
        return -1;        // не удалось создать
    }
    m_wndStatusBar.SetIndicators(indicators, sizeof(indicators)/sizeof(UINT));

    return 0;
}

```



```

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    if( !CFrameWnd::PreCreateWindow(cs) )
        return FALSE;
    // TODO: изменить класс Window или стили посредством изменения
    // CREATESTRUCT cs

    cs.style = WS_OVERLAPPED | WS_CAPTION | FWS_ADDTOTITLE
        | WS_THICKFRAME | WS_MINIMIZEBOX | WS_MAXIMIZEBOX;

    cs.dwExStyle &= ~WS_EX_CLIENTEDGE;
    cs.lpszClass = AfxRegisterWndClass(0);
    return TRUE;
}

// Диагностика CMainFrame

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
    CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
    CFrameWnd::Dump(dc);
}
#endif //_DEBUG

// Обработчики сообщений CMainFrame

void CMainFrame::OnSetFocus(CWnd* /*pOldWnd*/)
{
    // передача фокуса окну представления
    m_wndView.SetFocus();
}

BOOL CMainFrame::OnCmdMsg(UINT nID, int nCode, void* pExtra, AFX_CMDHANDLERINFO*
pHandlerInfo)
{
    // разрешить ошибки в представлении при выполнении команды
    if (m_wndView.OnCmdMsg(nID, nCode, pExtra, pHandlerInfo))
        return TRUE;

    // в противном случае выполняется обработка по умолчанию
    return CFrameWnd::OnCmdMsg(nID, nCode, pExtra, pHandlerInfo);
}

```

## LibGraph.h :

```

#ifndef LIBGRAPH
#define LIBGRAPH 1
const double pi = 3.14159;
typedef double(*pfunc2)(double, double); // Указатель на функцию

```

```

struct CSizeD
{
    double cx;
    double cy;
};
//-----
struct CRectD
{
    double left;
    double top;
    double right;
    double bottom;
    CRectD() { left = top = right = bottom = 0; };
    CRectD(double l, double t, double r, double b);
    void SetRectD(double l, double t, double r, double b);
    CSizeD SizeD();
};
//-----

CMatrix SpaceToWindow(CRectD& rs, CRect& rw);
// Возвращает матрицу пересчета координат из мировых в оконные
// rs - область в мировых координатах - double
// rw - область в оконных координатах - int
//-----
void SetMyMode(CDC& dc, CRect& RS, CRect& RW); //MFC
// Устанавливает режим отображения MM_ANISOTROPIC и его параметры
// dc - ссылка на класс CDC MFC
// RS - область в мировых координатах - int
// RW - область в оконных координатах - int
//-----
CMatrix CreateTranslate2D(double dx, double dy);
// Формирует матрицу для преобразования координат объекта при его смещении
// на dx по оси X и на dy по оси Y в фиксированной системе координат
// --- ИЛИ ---
// Формирует матрицу для преобразования координат объекта при смещении начала
// системы координат на -dx оси X и на -dy по оси Y при фиксированном положении
// объекта
//-----
CMatrix CreateTranslate3D(double dx, double dy, double dz);
// Формирует матрицу для преобразования координат объекта при его смещении
// на dx по оси X, на dy по оси Y, на dz по оси Z в фиксированной системе координат
// --- ИЛИ ---
// Формирует матрицу для преобразования координат объекта при смещении начала
// системы координат на -dx оси X, на -dy по оси Y, на -dz по оси Z
// при фиксированном положении объекта
//-----
CMatrix CreateRotate2D(double fi);
// Формирует матрицу для преобразования координат объекта при его повороте
// на угол fi (при fi>0 против часовой стрелки) в фиксированной системе координат
// --- ИЛИ ---
// Формирует матрицу для преобразования координат объекта при повороте начала
// системы координат на угол -fi при фиксированном положении объекта
// fi - угол в градусах
//-----
CMatrix CreateRotate3DZ(double fi);
// Формирует матрицу для преобразования координат объекта при его повороте вокруг оси
// Z
// на угол fi (при fi>0 против часовой стрелки) в фиксированной системе координат
// --- ИЛИ ---
// Формирует матрицу для преобразования координат объекта при повороте начала

```

```

// системы координат вокруг оси Z на угол -fi при фиксированном положении объекта
// fi - угол в градусах

//-----
CMatrix CreateRotate3DX(double fi);
// Формирует матрицу для преобразования координат объекта при его повороте вокруг оси
X
// на угол fi (при fi>0 против часовой стрелки) в фиксированной системе координат
// --- ИЛИ ---
// Формирует матрицу для преобразования координат объекта при повороте начала
// системы координат вокруг оси X на угол -fi при фиксированном положении объекта
// fi - угол в градусах

//-----
CMatrix CreateRotate3DY(double fi);
// Формирует матрицу для преобразования координат объекта при его повороте вокруг оси
Y
// на угол fi (при fi>0 против часовой стрелки) в фиксированной системе координат
// --- ИЛИ ---
// Формирует матрицу для преобразования координат объекта при повороте начала
// системы координат вокруг оси Y на угол -fi при фиксированном положении объекта
// fi - угол в градусах

//-----
CMatrix CreateViewCoord(double r, double fi, double q);
// Создает матрицу пересчета точки из мировой системы координат в видовую
// (r,fi,q)- координата ТОЧКИ НАБЛЮДЕНИЯ(начало видовой системы координат)
// в мировой сферической системе координат( углы fi и q в градусах)
//-----
CMatrix VectorMult(CMatrix& V1, CMatrix& V2);
// Вычисляет векторное произведение векторов V1 и V2
//-----
double ScalarMult(CMatrix& V1, CMatrix& V2);
// Вычисляет скалярное произведение векторов V1 и V2
//-----
double ModVec(CMatrix& V);
// Вычисляет модуль вектора V
//-----
double CosV1V2(CMatrix& V1, CMatrix& V2);
// Вычисляет КОСИНУС угла между векторами V1 и V2
//-----
double AngleV1V2(CMatrix& V1, CMatrix& V2);
// Вычисляет угол между векторами V1 и V2 в градусах
//-----
CMatrix SphereToCart(CMatrix& PView);
// Преобразует сферические координаты PView точки в декартовы
// PView(0) - r
// PView(1) - fi - азимут(отсчет от оси X), град.
// PView(2) - q - угол(отсчет от оси Z), град.
// Результат: R(0)- x, R(1)- y, R(2)- z
//-----
void GetProjection(CRectD& RS, CMatrix& Data, CMatrix& PView, CRectD& PR);
// Вычисляет координаты проекции охватывающего фигуру параллелепипеда на
// плоскость XY в ВИДОВОЙ системе координат
// Data - матрица данных
// RS - область на плоскости XY, на которую опирается отображаемая поверхность
// PView - координаты точки наблюдения в мировой сферической системе координат
// PR - проекция
//-----

#endif

```

## LibGraph.cpp:

```
#include "stdafx.h"

CRectD::CRectD(double l, double t, double r, double b)
{
    left = l;
    top = t;
    right = r;
    bottom = b;
}

//-----
void CRectD::SetRectD(double l, double t, double r, double b)
{
    left = l;
    top = t;
    right = r;
    bottom = b;
}

//-----
CSizeD CRectD::SizeD()
{
    CSizeD cz;
    cz.cx = fabs(right - left); // Ширина прямоугольной области
    cz.cy = fabs(top - bottom); // Высота прямоугольной области
    return cz;
}

//-----
CMatrix SpaceToWindow(CRectD& RS, CRect& RW)
// Возвращает матрицу пересчета координат из мировых в оконные
// RS - область в мировых координатах - double
// RW - область в оконных координатах - int
{
    CMatrix M(3, 3);
    CSize sz = RW.Size(); // Размер области в ОКНЕ
    int dwx = sz.cx; // Ширина
    int dwy = sz.cy; // Высота
    CSizeD szd = RS.SizeD(); // Размер области в МИРОВЫХ координатах

    double dsx = szd.cx; // Ширина в мировых координатах
    double dsy = szd.cy; // Высота в мировых координатах

    double kx = (double)dwx / dsx; // Масштаб по x
    double ky = (double)dwy / dsy; // Масштаб по y

    M(0, 0) = kx; M(0, 1) = 0; M(0, 2) = (double)RW.left - kx * RS.left;

    M(1, 0) = 0; M(1, 1) = -ky; M(1, 2) = (double)RW.bottom + ky * RS.bottom;

    M(2, 0) = 0; M(2, 1) = 0; M(2, 2) = 1;
    return M;
}
```

```

//-----

void SetMyMode(CDC& dc, CRect& RS, CRect& RW) //MFC
// Устанавливает режим отображения MM_ANISOTROPIC и его параметры
// dc - ссылка на класс CDC MFC
// RS - область в мировых координатах - int
// RW -      Область в оконных координатах - int
{
    int dsx = RS.right - RS.left;
    int dsy = RS.top - RS.bottom;
    int xsL = RS.left;
    int ysL = RS.bottom;

    int dwx = RW.right - RW.left;
    int dwy = RW.bottom - RW.top;
    int xwL = RW.left;
    int ywH = RW.bottom;

    dc.SetMapMode(MM_ANISOTROPIC);
    dc.SetWindowExt(dsx, dsy);
    dc.SetViewportExt(dwx, -dwy);
    dc.SetWindowOrg(xsL, ysL);
    dc.SetViewportOrg(xwL, ywH);
}
//-----

CMatrix CreateTranslate2D(double dx, double dy)
// Формирует матрицу для преобразования координат объекта при его смещении
// на dx по оси X и на dy по оси Y в фиксированной системе координат
// --- ИЛИ ---
// Формирует матрицу для преобразования координат объекта при смещении начала
// системы координат на -dx оси X и на -dy по оси Y при фиксированном положении
// объекта
{
    CMatrix TM(3, 3);
    TM(0, 0) = 1; TM(0, 2) = dx;
    TM(1, 1) = 1; TM(1, 2) = dy;
    TM(2, 2) = 1;
    return TM;
}
//-----
-
CMatrix CreateTranslate3D(double dx, double dy, double dz)
// Формирует матрицу для преобразования координат объекта при его смещении
// на dx по оси X, на dy по оси Y, на dz по оси Z в фиксированной системе координат
// --- ИЛИ ---
// Формирует матрицу для преобразования координат объекта при смещении начала
// системы координат на -dx оси X, на -dy по оси Y, на -dz по оси Z
// при фиксированном положении объекта
{
    CMatrix TM(4, 4);

```

```

        for (int i = 0; i < 4; i++) TM(i, i) = 1;
        TM(0, 3) = dx;
        TM(1, 3) = dy;
        TM(2, 3) = dz;
        return TM;
    }

//-----
CMatrix CreateRotate2D(double fi)
// Формирует матрицу для преобразования координат объекта при его повороте
// на угол fi (при fi>0 против часовой стрелки) в фиксированной системе координат
// --- ИЛИ ---
// Формирует матрицу для преобразования координат объекта при повороте начала
// системы координат на угол -fi при фиксированном положении объекта
// fi - угол в градусах
{
    double fg = fmod(fi, 360.0);
    double ff = (fg / 180.0)*pi; // Перевод в радианы
    CMatrix RM(3, 3);
    RM(0, 0) = cos(ff); RM(0, 1) = -sin(ff);
    RM(1, 0) = sin(ff); RM(1, 1) = cos(ff);
    RM(2, 2) = 1;
    return RM;
}
//-----

CMatrix CreateRotate3DZ(double fi)
// Формирует матрицу для преобразования координат объекта при его повороте вокруг оси
// Z
// на угол fi (при fi>0 против часовой стрелки) в фиксированной системе координат
// --- ИЛИ ---
// Формирует матрицу для преобразования координат объекта при повороте начала
// системы координат вокруг оси Z на угол -fi при фиксированном положении объекта
// fi - угол в градусах
{
    double fg = fmod(fi, 360.0);
    double ff = (fg / 180.0)*pi; // Перевод в радианы
    CMatrix RM(4, 4);
    RM(0, 0) = cos(ff); RM(0, 1) = -sin(ff);
    RM(1, 0) = sin(ff); RM(1, 1) = cos(ff);
    RM(2, 2) = 1;
    RM(3, 3) = 1;
    return RM;
}
//-----

CMatrix CreateRotate3DX(double fi)
// Формирует матрицу для преобразования координат объекта при его повороте вокруг оси
// X
// на угол fi (при fi>0 против часовой стрелки) в фиксированной системе координат
// --- ИЛИ ---
// Формирует матрицу для преобразования координат объекта при повороте начала
// системы координат вокруг оси X на угол -fi при фиксированном положении объекта

```

```

// fi - угол в градусах
{
    double fg = fmod(fi, 360.0);
    double ff = (fg / 180.0)*pi; // Перевод в радианы
    CMatrix RM(4, 4);
    RM(0, 0) = 1;
    RM(1, 1) = cos(ff); RM(1, 2) = -sin(ff);
    RM(2, 1) = sin(ff); RM(2, 2) = cos(ff);
    RM(3, 3) = 1;
    return RM;
}

//-----
CMatrix CreateRotate3DY(double fi)
// Формирует матрицу для преобразования координат объекта при его повороте вокруг оси
Y
// на угол fi (при fi>0 против часовой стрелки) в фиксированной системе координат
// --- ИЛИ ---
// Формирует матрицу для преобразования координат объекта при повороте начала
// системы координат вокруг оси Y на угол -fi при фиксированном положении объекта
// fi - угол в градусах
{
    double fg = fmod(fi, 360.0);
    double ff = (fg / 180.0)*pi; // Перевод в радианы
    CMatrix RM(4, 4);
    RM(0, 0) = cos(ff); RM(0, 2) = sin(ff);
    RM(1, 1) = 1;
    RM(2, 0) = -sin(ff); RM(2, 2) = cos(ff);
    RM(3, 3) = 1;
    return RM;
}

//-----
CMatrix VectorMult(CMatrix& V1, CMatrix& V2)
// Вычисляет векторное произведение векторов V1 и V2
{
    int b1 = (V1.cols() == 1) && (V1.rows() == 3);
    int b2 = (V2.cols() == 1) && (V2.rows() == 3);
    int b = b1 && b2;
    if (!b)
    {
        //char* error="VectorMult: неправильные размерности векторов! ";
        const TCHAR error[] = _T("VectorMult: неправильные размерности векторов!");
    );
        MessageBox(NULL, error, _T("Ошибка"), MB_ICONSTOP);
        exit(1);
    }
    CMatrix W(3);
    W(0) = V1(1)*V2(2) - V1(2)*V2(1);
    //double x=W(0);
    W(1) = -(V1(0)*V2(2) - V1(2)*V2(0));
    //double y=W(1);
    W(2) = V1(0)*V2(1) - V1(1)*V2(0);
    //double z=W(2);
}

```

```

        return W;
    }

//-----
double ScalarMult(CMatrix& V1, CMatrix& V2)
// Вычисляет скалярное произведение векторов V1 и V2
{
    int b1 = (V1.cols() == 1) && (V1.rows() == 3);
    int b2 = (V2.cols() == 1) && (V2.rows() == 3);
    int b = b1 && b2;
    if (!b)
    {
        char* error = "ScalarMult: неправильные размерности векторов! ";
        //MessageBox(NULL, error, "Ошибка", MB_ICONSTOP);
        exit(1);
    }
    double p = V1(0)*V2(0) + V1(1)*V2(1) + V1(2)*V2(2);
    return p;
}

//-----
double ModVec(CMatrix& V)
// Вычисляет модуль вектора V
{
    int b = (V.cols() == 1) && (V.rows() == 3);
    if (!b)
    {
        char* error = "ModVec: неправильная размерность вектора! ";
        //MessageBox(NULL, error, "Ошибка", MB_ICONSTOP);
        exit(1);
    }
    double q = sqrt(V(0)*V(0) + V(1)*V(1) + V(2)*V(2));
    return q;
}

//-----
double CosV1V2(CMatrix& V1, CMatrix& V2)
// Вычисляет КОСИНУС угла между векторами V1 и V2
{
    double modV1 = ModVec(V1);
    double modV2 = ModVec(V2);
    int b = (modV1 < 1e-7) || (modV2 < 1e-7);
    if (b)
    {
        char* error = "CosV1V2: модуль одного или обоих векторов < 1e-7!";
        //MessageBox(NULL, error, "Ошибка", MB_ICONSTOP);
        exit(1);
    }
    int b1 = (V1.cols() == 1) && (V1.rows() == 3);
    int b2 = (V2.cols() == 1) && (V2.rows() == 3);
    b = b1 && b2;
    if (!b)
    {
        char* error = "CosV1V2: неправильные размерности векторов! ";
        //MessageBox(NULL, error, "Ошибка", MB_ICONSTOP);
        exit(1);
    }
}

```



```

    }
    double cos_f = ScalarMult(V1, V2) / (modV1*modV2);
    return cos_f;
}

//-----
double AngleV1V2(CMatrix& V1, CMatrix& V2)
// Вычисляет угол между векторами V1 и V2 в градусах
{
    double modV1 = ModVec(V1);
    double modV2 = ModVec(V2);
    int b = (modV1 < 1e-7) || (modV2 < 1e-7);
    if (!b)
    {
        char* error = "AngleV1V2: модуль одного или обоих векторов < 1e-7!";
        //MessageBox(NULL, error, "Ошибка", MB_ICONSTOP);
        exit(1);
    }
    int b1 = (V1.cols() == 1) && (V1.rows() == 3);
    int b2 = (V2.cols() == 1) && (V2.rows() == 3);
    b = b1 && b2;
    if (!b)
    {
        char* error = "AngleV1V2: неправильные размерности векторов! ";
        //MessageBox(NULL, error, "Ошибка", MB_ICONSTOP);
        exit(1);
    }
    double cos_f = ScalarMult(V1, V2) / (modV1*modV2);
    if (fabs(cos_f) > 1)
    {
        char* error = "AngleV1V2: модуль cos(f)>1! ";
        //MessageBox(NULL, error, "Ошибка", MB_ICONSTOP);
        exit(1);
    }
    double f;
    if (cos_f > 0) f = acos(cos_f);
    else f = pi - acos(cos_f);
    double fg = (f / pi) * 180;
    return fg;
}

//-----
CMatrix CreateViewCoord(double r, double fi, double q)
// Создает матрицу пересчета (4x4) точки из мировой системы координат в видовую
// (r,fi,q)- координата ТОЧКИ НАБЛЮДЕНИЯ(начало видовой системы координат)
// в мировой сферической системе координат( углы fi и q в градусах)
{
    double fg = fmod(fi, 360.0);
    double ff = (fg / 180.0)*pi; // Перевод в радианы
    fg = fmod(q, 360.0);
    double qq = (fg / 180.0)*pi; // Перевод в радианы

    CMatrix VM(4, 4);
    VM(0, 0) = -sin(ff);    VM(0, 1) = cos(ff);

```

```

        VM(1, 0) = -cos(qq)*cos(ff);        VM(1, 1) = -cos(qq)*sin(ff);        VM(1, 2) =
sin(qq);
        VM(2, 0) = -sin(qq)*cos(ff);        VM(2, 1) = -sin(qq)*sin(ff);        VM(2, 2) =
-cos(qq); VM(2, 3) = r;
        VM(3, 3) = 1;
        return VM;
}

//-----
CMatrix SphereToCart(CMatrix& PView)
// Преобразует сферические координаты PView точки в декартовы
// PView(0) - r
// PView(1) - fi - азимут(отсчет от оси X), град.
// PView(2) - q - угол(отсчет от оси Z), град.
// Результат: R(0)- x, R(1)- y, R(2)- z
{
    CMatrix R(3);
    double r = PView(0);
    double fi = PView(1);
    double q = PView(2);
    double fi_rad = (fi / 180.0)*pi; // Перевод fi в радианы
    double q_rad = (q / 180.0)*pi; // Перевод q в радианы
    R(0) = r * sin(q_rad)*cos(fi_rad); // x- координата точки наблюдения
    R(1) = r * sin(q_rad)*sin(fi_rad); // y- координата точки наблюдения
    R(2) = r * cos(q_rad); // z-
координата точки наблюдения
    return R;
}

//----- GetProjection -----

void GetProjection(CRectD& RS, CMatrix& Data, CMatrix& PView, CRectD& PR)
// Вычисляет координаты проекции охватывающего фигуру параллелепипеда на
// плоскость XY в ВИДОВОЙ системе координат
// Data - матрица данных
// RS - область на плоскости XY, на которую опирается отображаемая поверхность
// PView - координаты точки наблюдения в мировой сферической системе координат
// PR - проекция
{
    double Zmax = Data.MaxElement();
    double Zmin = Data.MinElement();
    CMatrix PS(4, 4); // Точки в мировом пространстве
    PS(3, 0) = 1; PS(3, 1) = 1; PS(3, 2) = 1; PS(3, 3) = 1;
    CMatrix MV = CreateViewCoord(PView(0), PView(1), PView(2)); //Матрица(4x4)
пересчета
    //в видовую систему координат
    PS(0, 0) = RS.left;
    PS(1, 0) = RS.top;
    PS(2, 0) = Zmax;

    PS(0, 1) = RS.left;
    PS(1, 1) = RS.bottom;
    PS(2, 1) = Zmax;

```

```

    PS(0, 2) = RS.right;
    PS(1, 2) = RS.top;
    PS(2, 2) = Zmax;

    PS(0, 3) = RS.right;
    PS(1, 3) = RS.bottom;
    PS(2, 3) = Zmax;

    CMatrix Q = MV * PS;          // Координаты верхней плоскости параллелепипеда в
видовой СК
    CMatrix V = Q.GetRow(0);       // Строка X - координат
    double Xmin = V.MinElement();
    double Xmax = V.MaxElement();
    V = Q.GetRow(1);              // Строка Y - координат
    double Ymax = V.MaxElement();

    PS(2, 0) = Zmin;
    PS(2, 1) = Zmin;
    PS(2, 2) = Zmin;
    PS(2, 3) = Zmin;

    Q = MV * PS;                  // Координаты нижней плоскости параллелепипеда в
видовой СК
    V = Q.GetRow(1);              // Строка Y - координат
    double Ymin = V.MinElement();
    PR.SetRectD(Xmin, Ymax, Xmax, Ymin);
}

```

## LibSurface.h :

```

#pragma once

#ifndef LIBSURFACE
#define LIBSURFACE 1

using namespace std;

////////// === ОПРЕДЕЛЕНИЕ ТИПОВ === //////////

typedef vector<CMatrix> CVecMatrix;
typedef vector<CVecMatrix> CMasMatrix;

typedef vector<CPoint> CVecPoint;
typedef vector<CVecPoint> CMatrPoint;

//----- Тестовая функция -----

double Function1(double x, double y);
double Function2(double x, double y);
double Function3(double x, double y);

//----- class CPlot3D -----
class CPlot3D
{

```

```

    pfunc2 pFunc;           // Указатель на функцию f(x,y), описывающую
поверхность
    CRectD SpaceRect;       // Прямоугольная область, на которую опирается
поверхность в мировой СК
    CMasMatrix MatrF;       // Матрица для хранения координат точек (x,y,z,1)
поверхности в мировой СК
    CMasMatrix MatrView;    // Матрица для хранения координат точек (x,y,1) проекции
поверхности на плоскость XY видовой СК
    CRectD ViewRect;        // Прямоугольная область, охватывающая проекцию
поверхности на плоскость XY видовой СК
    CRect WinRect;          // Прямоугольная область в оконной системе
координат для рисования
    CMatrix ViewPoint;      // Вектор (3x1) - координаты точки наблюдения в
мировой
                                // сферической системе координат
(r,fi(град.), q(град.))
    CMatrPoint MatrWindow;  // Матрица для хранения оконных координат P(xi,yi)
точек изображения

public:
    CPlot3D();
    ~CPlot3D() { pFunc = NULL; };
    void SetFunction(pfunc2 pF, CRectD RS, double dx, double dy); // Устанавливает
функцию f(x,y)
    void SetViewPoint(double r, double fi, double q);             // Устанавливает
положение точки наблюдения в МИРОВОЙ СК
    CMatrix GetViewPoint();                                       // Возвращает вектор ViewPoin
    void SetWinRect(CRect Rect);                                  // Устанавливает область в окне для
рисования
    void CreateMatrF(double dx, double dy); // Заполняет матрицу MatrF координатами
точек поверхности
    void CreateMatrView();                                       // Заполняет матрицу MatrView координатами
точек проекции поверхности на плоскость XY видовой СК
    void CreateMatrWindow();  // Заполняет матрицу MatrWindow
    void Draw(CDC& dc);
};

#endif

```

## LibSurface.cpp:

```

#include "stdafx.h"

double Function1(double x, double y)/////////
{
    return x * x + y * y;
};

double Function2(double x, double y)/////////
{
    return x * x - y * y;
};

double Function3(double x, double y)/////////
{
    if (y != 0 && x != 0)
        if ((1 - x * x - y * y) >= 0)
            return sqrt(1 - x * x - y * y);
};

CPlot3D::CPlot3D()/////////
{

```

```

    pFunc = NULL;
    ViewPoint.RedimMatrix(3);
    WinRect.SetRect(0, 0, 200, 200);
    ViewPoint(0) = 10, ViewPoint(1) = 30;
    ViewPoint(2) = 45;
};

void CPlot3D::SetFunction(pfunc2 pF, CRectD RS, double dx, double dy)/////////
{
    pFunc = pF;
    SpaceRect.SetRectD(RS.left, RS.top, RS.right, RS.bottom);
    MatrF.clear(); // очищаем матрицу для хранения координат точек
поверхности в МСК
    MatrView.clear(); // очищаем матрицу для хранения к-т точек проекции
поверхности на плоскость XY ВСК
    MatrWindow.clear(); // очищаем матрицу для хранения оконных координат точек
изображения

    CreateMatrF(dx, dy); // заполняем матрицу к-ми точек поверхности
    CreateMatrView(); // заполняем матрицу к-ми точек проекции поверхности на
плоскость XY ВСК
    CreateMatrWindow(); // заполняем матрицу
}

void CPlot3D::SetViewPoint(double r, double fi, double q)/////////
{
    ViewPoint(0) = r, ViewPoint(1) = fi;
    ViewPoint(2) = q;
    MatrView.clear();
    CreateMatrView();
    MatrWindow.clear();
    CreateMatrWindow();
}

CMatrix CPlot3D::GetViewPoint()
{
    CMatrix P = ViewPoint;
    return P;
}

void CPlot3D::SetWinRect(CRect Rect)/////////
{
    WinRect = Rect;
    MatrWindow.clear();
    CreateMatrWindow();
}

void CPlot3D::CreateMatrF(double dx, double dy)/////////
{
    double xL = SpaceRect.left;
    double xH = SpaceRect.right;
    double yL = SpaceRect.bottom;
    double yH = SpaceRect.top;
    CVecMatrix VecMatrix;
    CMatrix V(4);
    V(3) = 1;
    for (double x = xL; x <= xH; x += dx)
    {
        VecMatrix.clear();
        for (double y = yL; y <= yH; y += dy)
        {
            V(0) = x;

```

```

        V(1) = y;
        V(2) = pFunc(x, y);
        VecMatrix.push_back(V);
    }
    MatrF.push_back(VecMatrix);
}

}

void CPlot3D::CreateMatrView()//////////
{
    CMatrix MV = CreateViewCoord(ViewPoint(0), ViewPoint(1), ViewPoint(2));
    CVecMatrix VecMatrix;
    CMatrix VX(4), V(3);
    V(2) = 1;
    double xmin = DBL_MAX;
    double xmax = DBL_MIN;
    double ymin = DBL_MAX;
    double ymax = DBL_MIN;

    for (int i = 0; i < MatrF.size(); i++)
    {
        VecMatrix.clear();
        for (int j = 0; j < MatrF[i].size(); j++)
        {
            VX = MatrF[i][j];
            VX = MV * VX;
            V(0) = VX(0);
            V(1) = VX(1);
            VecMatrix.push_back(V);

            double x = V(0);
            double y = V(1);
            if (x < xmin) xmin = x;
            if (x > xmax) xmax = x;
            if (y < ymin) ymin = y;
            if (y > ymax) ymax = y;
        }
        MatrView.push_back(VecMatrix);
    }
    ViewRect.SetRectD(xmin, ymax, xmax, ymin);
}

void CPlot3D::CreateMatrWindow()//////////
{
    CMatrix MW = SpaceToWindow(ViewRect, WinRect);
    CVecPoint VecPoint;
    CMatrix V(3);
    for (int i = 0; i < MatrView.size(); i++)
    {
        VecPoint.clear();
        for (int j = 0; j < MatrView[i].size(); j++)
        {
            V = MatrView[i][j];
            V = MW * V;
            CPoint P((int)V(0), (int)V(1));
            VecPoint.push_back(P);
        }
        MatrWindow.push_back(VecPoint);
    }
}

void CPlot3D::Draw(CDC& dc)//////////

```

```

{
    if (MatrWindow.empty())
    {
        TCHAR* error = _T("Массив данных для рисования в окне пуст! ");
        MessageBox(NULL, error, _T("Ошибка"), MB_ICONSTOP);
        return;
    }
    CPoint pt[4];
    int k, d;
    int nRows = MatrWindow.size();
    int nCols = MatrWindow[0].size();
    for (int i = 0; i < nRows - 1; i++)
        for (int j = 0; j < nCols - 1; j++)
        {
            pt[0] = MatrWindow[i][j];
            pt[1] = MatrWindow[i][j + 1];
            pt[2] = MatrWindow[i + 1][j + 1];
            pt[3] = MatrWindow[i + 1][j];
            dc.Polygon(pt, 4);
        }
}

```

Далее нужно реализовать функции для создания матриц и расчета координат наших функций, файлы ChildView.h и CMatrix.h. И осуществить вывод этих функций в окно приложения.

#### ChildView.h :

```

// ChildView.h: интерфейс класса CChildView
//

#pragma once

// Окно CChildView

class CChildView : public CWnd
{
// Создание
public:
    CChildView();

// Атрибуты
public:
    double r, fi, q; // Сферические координаты точки наблюдения в МСК
    CRect WRect;
    int Index; // Индикатор для OnPaint
    CPlot3D Graph1, Graph2, Graph3; // Объявление объектов типа CPlot3D
// Операции
public:

// Переопределение
protected:
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);

// Реализация
public:
    virtual ~CChildView();

```

```

        // Созданные функции схемы сообщений
protected:
    afx_msg void OnPaint();
    DECLARE_MESSAGE_MAP()
public:
    // действия при выборе пункта меню
    afx_msg void OnCPlot3DDef();
    afx_msg void OnCplot3dFunc1();
    afx_msg void OnCplot3dFunc2();
    afx_msg void OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags);
};

```

## ChildView.cpp:

```

// ChildView.cpp: реализация класса CChildView
//

#include "stdafx.h"
#include "Lab06.h"
#include "ChildView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CChildView

CChildView::CChildView()
{
    Index = 0;    // Индикатор для OnPaint
}

CChildView::~CChildView()
{
}

// Реализация карты сообщений
BEGIN_MESSAGE_MAP(CChildView, CWnd)
    ON_WM_PAINT()
    // сообщения меню выбора
    ON_COMMAND(ID_CPlot3D_FUNC1, &CChildView::OnCPlot3DDef)
    ON_COMMAND(ID_CPlot3D_FUNC2, &CChildView::OnCplot3dFunc1)
    ON_COMMAND(ID_CPlot3D_FUNC3, &CChildView::OnCplot3dFunc2)
    ON_WM_KEYDOWN()
END_MESSAGE_MAP()

// Обработчики сообщений CChildView

BOOL CChildView::PreCreateWindow(CREATESTRUCT& cs)
{
    if (!CWnd::PreCreateWindow(cs))
        return FALSE;

    cs.dwExStyle |= WS_EX_CLIENTEDGE;
    cs.style &= ~WS_BORDER;
    cs.lpszClass = AfxRegisterWndClass(CS_HREDRAW|CS_VREDRAW|CS_DBLCLKS,
        ::LoadCursor(nullptr, IDC_ARROW),
        reinterpret_cast<HBRUSH>(COLOR_WINDOW+1), nullptr);
}

```



```

        return TRUE;
    }

void CChildView::OnPaint()
{
    CPaintDC dc(this); // контекст устройства для рисования

    CString ss;

    if (Index == 1) Graph1.Draw(dc);
    if (Index == 2) Graph2.Draw(dc);
    if (Index == 3) Graph3.Draw(dc);
    if (Index > 0)
    {
        ss.Format(L"f=%5.1f, q=%5.1f", fi, q);
        dc.TextOutW(5, 5, ss);
    }
}

void CChildView::OnCPlot3DDef() // функция Function1
{
    double dx = 0.25, dy = 0.25; // шаг для рисования
    r = 50, fi = 30, q = 45; // сферические координаты положения
    камеры (наблюдателя)
    CRectD SpaceRect(-5, 5, 5, -5); // область для рисования
    CRect WinRect;
    this->GetClientRect(WinRect); // получаем координаты окна
    WinRect.SetRect(WinRect.left + 50, WinRect.top + 50, WinRect.right - 50,
WinRect.bottom - 50); // устанавливаем координаты окна
    Graph1.SetFunction(Function1, SpaceRect, dx, dy); // устанавливаем
Function1, область рисования и шаг
    Graph1.SetViewPoint(r, fi, q); // устанавливаем положение наблюдателя
    Graph1.SetWinRect(WinRect); // устанавливаем область в окне для
рисования
    Index = 1;
    this->Invalidate();
}

void CChildView::OnCplot3dFunc1() // функция Function2
{
    double dx = 0.25, dy = 0.25;
    r = 50, fi = 30, q = 45;
    CRectD SpaceRect(-5, 5, 5, -5);
    CRect WinRect;
    this->GetClientRect(WinRect);
    WinRect.SetRect(WinRect.left + 50, WinRect.top + 50, WinRect.right - 50,
WinRect.bottom - 50);
    Graph2.SetFunction(Function2, SpaceRect, dx, dy);
    Graph2.SetViewPoint(r, fi, q);
    Graph2.SetWinRect(WinRect);
    Index = 2;
    this->Invalidate();
}

void CChildView::OnCplot3dFunc2() // Функция Function3
{
    double dx = 0.2, dy = 0.2;
    r = 50, fi = 30, q = 45;
    CRectD SpaceRect(-5, 5, 5, -5);
    CRect WinRect;
    this->GetClientRect(WinRect);

```

```

        WinRect.SetRect(WinRect.left + 50, WinRect.top + 50, WinRect.right - 50,
WinRect.bottom - 50);
        Graph3.SetFunction(Function3, SpaceRect, dx, dy);           // Function3
        Graph3.SetViewPoint(r, fi, q);
        Graph3.SetWinRect(WinRect);
        Index = 3;
        this->Invalidate();
    }

void CChildView::OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)
{
    CMatrix P(3);
    switch (Index)
    {
        case 1:
        {
            P = Graph1.GetViewPoint();
            break;
        }
        case 2:
        {
            P = Graph2.GetViewPoint();
            break;
        }
        case 3:
        {
            P = Graph3.GetViewPoint();
            break;
        }
    }

    r = P(0), fi = P(1), q = P(2);
    double delta_fi = 5, delta_q = 5;
    switch (nChar)
    {
        case VK_UP:
        {
            double qx = q - delta_q;
            if (qx >= 0)q = qx;
            break;
        }
        case VK_DOWN:
        {
            double qx = q + delta_q;
            if (qx <= 180)q = qx;
            break;
        }
        case VK_LEFT:
        {
            double fix = fi - delta_fi;
            if (fix >= 0)fi = fix;
            break;
        }
        case VK_RIGHT:
        {
            double fix = fi + delta_fi;
            if (fix <= 360)fi = fix;
            break;
        }
    }

    switch (Index)

```

```

{
    case 1:
    {
        Graph1.SetViewPoint(r, fi, q);
        break;
    }

    case 2:
    {
        Graph2.SetViewPoint(r, fi, q);
        break;
    }

    case 3:
    {
        Graph3.SetViewPoint(r, fi, q);
        break;
    }
}
CWnd::OnKeyDown(nChar, nRepCnt, nFlags);
this->Invalidate();
}

```

## CMatrix.h :

```

#pragma once
#include <fstream>
using namespace std;
#ifndef CMATRIXH
# define CMATRIXH 1
class CMatrix
{
    double **array;
    int n_rows; // Число строк
    int n_cols; // Число столбцов
public:
    CMatrix(); // Конструктор по
    // умолчанию (1 на 1)
    CMatrix(int, int); // Конструктор
    CMatrix(int); // Конструктор -вектора (один
    // столбец)
    CMatrix(const CMatrix&); // Конструктор копирования
    CMatrix(ifstream &file)
    {
        int r = 0;
        int c = 0;
        file >> r;
        file >> c;

        // Nrow - число строк
        // Ncol - число столбцов

        n_rows = r;
        n_cols = c;
        array = new double*[n_rows];
        for (int i = 0; i < n_rows; i++) array[i] = new double[n_cols];

        for (int i = 0; i < n_rows; i++)
            for (int j = 0; j < n_cols; j++) file >> array[i][j];
    }
}

```

```

~CMatrix();
double &operator()(int, int);           // Выбор элемента матрицы по индексу
double &operator()(int);                // Выбор элемента вектора по индексу
CMatrix operator-();                    // Оператор "-"
CMatrix operator=(const CMatrix&);      // Оператор "Присвоить": M1=M2
CMatrix operator*(CMatrix&);           // Оператор "Произведение": M1*M2
CMatrix operator+(CMatrix&);           // Оператор "+": M1+M2
CMatrix operator-(CMatrix&);           // Оператор "-": M1-M2
CMatrix operator+(double);             // Оператор "+": M+a
CMatrix operator-(double);             // Оператор "-": M-a
friend std::ostream& operator<<(std::ostream& os, CMatrix& matrix)
{
    os << matrix.rows() << ' ' << matrix.cols() << '\n';

    for (int i = 0; i < matrix.rows(); i++)
    {
        for (int k = 0; k < matrix.cols(); k++)
        {
            os << matrix(i, k) << " ";
        }
        os << '\n';
    }
    return os;
}
int rows()const { return n_rows; };    // Возвращает число строк
int cols()const { return n_cols; };    // Возвращает число строк
CMatrix Transp();                      // Возвращает
матрицу, транспонированную к текущей
CMatrix GetRow(int);                  // Возвращает строку по номеру
CMatrix GetRow(int, int, int);        // Возвращает строку по номеру
CMatrix GetCol(int);                  // Возвращает столбец по номеру
CMatrix GetCol(int, int, int);        // Возвращает столбец по номеру
CMatrix RedimMatrix(int, int);        // Изменяет размер матрицы с уничтожением
данных
CMatrix RedimData(int, int);          // Изменяет размер матрицы с сохранением
данных,                                     //которые можно
сохранить
CMatrix RedimMatrix(int);             // Изменяет размер матрицы с уничтожением
данных
CMatrix RedimData(int);               // Изменяет размер матрицы с сохранением
данных,                                     //которые можно
сохранить
double MaxElement();                  // Максимальный элемент матрицы
double MinElement();                  // Минимальный элемент матрицы
};

#endif

```

## CMatrix.cpp:

```

#include "stdafx.h"
#include "CMatrix.h"

CMatrix::CMatrix()
{
    n_rows = 1;
    n_cols = 1;
    array = new double*[n_rows];
}

```

```

        for (int i = 0; i < n_rows; i++) array[i] = new double[n_cols];
        for (int i = 0; i < n_rows; i++)
            for (int j = 0; j < n_cols; j++) array[i][j] = 0;
    }

//-----
CMatrix::CMatrix(int Nrow, int Ncol)
// Nrow - число строк
// Ncol - число столбцов
{
    n_rows = Nrow;
    n_cols = Ncol;
    array = new double*[n_rows];
    for (int i = 0; i < n_rows; i++) array[i] = new double[n_cols];
    for (int i = 0; i < n_rows; i++)
        for (int j = 0; j < n_cols; j++) array[i][j] = 0;
}

//-----
CMatrix::CMatrix(int Nrow) //Вектор
// Nrow - число строк
{
    n_rows = Nrow;
    n_cols = 1;
    array = new double*[n_rows];
    for (int i = 0; i < n_rows; i++) array[i] = new double[n_cols];
    for (int i = 0; i < n_rows; i++)
        for (int j = 0; j < n_cols; j++) array[i][j] = 0;
}

//-----
CMatrix::~CMatrix()
{
    for (int i = 0; i < n_rows; i++) delete array[i];
    delete array;
}

//-----
double &CMatrix::operator()(int i, int j)
// i - номер строки
// j - номер столбца
{
    if ((i > n_rows - 1) || (j > n_cols - 1)) // проверка выхода за диапазон
    {
        TCHAR* error = _T("CMatrix::operator(int,int): выход индекса за границу
диапазона ");
        MessageBox(NULL, error, _T("Ошибка"), MB_ICONSTOP);
        exit(1);
    }
    return array[i][j];
}

//-----
double &CMatrix::operator()(int i)
// i - номер строки для вектора

```

```

{
    if (n_cols > 1)        // Число столбцов больше одного
    {
        char* error = "CMatrix::operator(int): объект не вектор - число столбцов
        больше 1 ";
        MessageBox(NULL, (LPCWSTR)error, L"Ошибка", MB_ICONSTOP);
        exit(1);
    }
    if (i > n_rows - 1)    // проверка выхода за диапазон
    {
        TCHAR* error = TEXT("CMatrix::operator(int): выход индекса за границу
        диапазона ");
        MessageBox(NULL, error, TEXT("Ошибка"), MB_ICONSTOP);
        exit(1);
    }
    return array[i][0];
}

//-----
CMatrix CMatrix::operator-()
// Оператор -M
{
    CMatrix Temp(n_rows, n_cols);
    for (int i = 0; i < n_rows; i++)
        for (int j = 0; j < n_cols; j++) Temp(i, j) = -array[i][j];
    return Temp;
}

//-----
CMatrix CMatrix::operator+(CMatrix& M)
// Оператор M1+M2
{
    int bb = (n_rows == M.rows()) && (n_cols == M.cols());
    if (!bb)
    {
        char* error = "CMatrix::operator(+): несоответствие размерностей матриц
        ";
        MessageBox(NULL, (LPCWSTR)error, L"Ошибка", MB_ICONSTOP);
        exit(1);
    }
    CMatrix Temp(*this);
    for (int i = 0; i < n_rows; i++)
        for (int j = 0; j < n_cols; j++) Temp(i, j) += M(i, j);
    return Temp;
}

//-----
CMatrix CMatrix::operator-(CMatrix& M)
// Оператор M1-M2
{
    int bb = (n_rows == M.rows()) && (n_cols == M.cols());
    if (!bb)
    {
        char* error = "CMatrix::operator(-): несоответствие размерностей матриц
        ";

```

```

        MessageBox(NULL, (LPCWSTR)error, L"Ошибка", MB_ICONSTOP);
        exit(1);
    }
    CMatrix Temp(*this);
    for (int i = 0; i < n_rows; i++)
        for (int j = 0; j < n_cols; j++) Temp(i, j) -= M(i, j);
    return Temp;
}
//-----
CMatrix CMatrix::operator*(CMatrix& M)
// Умножение на матрицу M
{
    double sum;
    int nn = M.rows();
    int mm = M.cols();
    CMatrix Temp(n_rows, mm);
    if (n_cols == nn)
    {
        for (int i = 0; i < n_rows; i++)
            for (int j = 0; j < mm; j++)
            {
                sum = 0;
                for (int k = 0; k < n_cols; k++) sum += (*this)(i, k)*M(k,
j));
                Temp(i, j) = sum;
            }
    }
    else
    {
        TCHAR* error = TEXT("CMatrix::operator*: несоответствие размерностей
матриц ");
        MessageBox(NULL, error, TEXT("Ошибка"), MB_ICONSTOP);
        exit(1);
    }
    return Temp;
}
//-----
CMatrix CMatrix::operator=(const CMatrix& M)
// Оператор присваивания M1=M
{
    if (this == &M) return *this;
    int nn = M.rows();
    int mm = M.cols();
    if ((n_rows == nn) && (n_cols == mm))
    {
        for (int i = 0; i < n_rows; i++)
            for (int j = 0; j < n_cols; j++) array[i][j] = M.array[i][j];
    }
    else // для ошибки размерностей
    {
        TCHAR* error = TEXT("CMatrix::operator=: несоответствие размерностей
матриц");
        MessageBox(NULL, error, TEXT("Ошибка"), MB_ICONSTOP);
    }
}

```

```

        exit(1);
    }
    return *this;
}

//-----
CMatrix::CMatrix(const CMatrix &M) // Конструктор копирования
{
    n_rows = M.n_rows;
    n_cols = M.n_cols;
    array = new double*[n_rows];
    for (int i = 0; i < n_rows; i++) array[i] = new double[n_cols];
    for (int i = 0; i < n_rows; i++)
        for (int j = 0; j < n_cols; j++) array[i][j] = M.array[i][j];
}

//-----
CMatrix CMatrix::operator+(double x)
// Оператор M+x, где M - матрица, x - число
{
    CMatrix Temp(*this);
    for (int i = 0; i < n_rows; i++)
        for (int j = 0; j < n_cols; j++) Temp(i, j) += x;
    return Temp;
}

//-----
CMatrix CMatrix::operator-(double x)
// Оператор M-x, где M - матрица, x - число
{
    CMatrix Temp(*this);
    for (int i = 0; i < n_rows; i++)
        for (int j = 0; j < n_cols; j++) Temp(i, j) -= x;
    return Temp;
}

//-----
CMatrix CMatrix::Transp()
// Возвращает матрицу, транспонированную к (*this)
{
    CMatrix Temp(n_cols, n_rows);
    for (int i = 0; i < n_cols; i++)
        for (int j = 0; j < n_rows; j++) Temp(i, j) = array[j][i];
    return Temp;
}

//-----
CMatrix CMatrix::GetRow(int k)
// Возвращает строку матрицы по номеру k
{
    if (k > n_rows - 1)
    {

```



```

        char* error = "CMatrix::GetRow(int k): параметр k превышает число строк
";
        MessageBox(NULL, (LPCWSTR)error, L"Ошибка", MB_ICONSTOP);
        exit(1);
    }
    CMatrix M(1, n_cols);
    for (int i = 0; i < n_cols; i++)M(0, i) = (*this)(k, i);
    return M;
}

//-----
CMatrix CMatrix::GetRow(int k, int n, int m)
// Возвращает подстроку из строки матрицы с номером k
// n - номер первого элемента в строке
// m - номер последнего элемента в строке
{
    int b1 = (k >= 0) && (k < n_rows);
    int b2 = (n >= 0) && (n <= m);
    int b3 = (m >= 0) && (m < n_cols);
    int b4 = b1 && b2&&b3;
    if (!b4)
    {
        char* error = "CMatrix::GetRow(int k,int n, int m):ошибка в параметрах
";
        MessageBox(NULL, (LPCWSTR)error, L"Ошибка", MB_ICONSTOP);
        exit(1);
    }
    int nCols = m - n + 1;
    CMatrix M(1, nCols);
    for (int i = n; i <= m; i++)M(0, i - n) = (*this)(k, i);
    return M;
}

//-----
CMatrix CMatrix::GetCol(int k)
// Возвращает столбец матрицы по номеру k
{
    if (k > n_cols - 1)
    {
        char* error = "CMatrix::GetCol(int k): параметр k превышает число
столбцов ";
        MessageBox(NULL, (LPCWSTR)error, L"Ошибка", MB_ICONSTOP);
        exit(1);
    }
    CMatrix M(n_rows, 1);
    for (int i = 0; i < n_rows; i++)M(i, 0) = (*this)(i, k);
    return M;
}

//-----
CMatrix CMatrix::GetCol(int k, int n, int m)
// Возвращает подстолбец из столбца матрицы с номером k
// n - номер первого элемента в столбце
// m - номер последнего элемента в столбце
{
    int b1 = (k >= 0) && (k < n_cols);

```

```

        int b2 = (n >= 0) && (n <= m);
        int b3 = (m >= 0) && (m < n_rows);
        int b4 = b1 && b2&&b3;
        if (!b4)
        {
            char* error = "CMatrix::GetCol(int k,int n, int m):ошибка в параметрах
";
            MessageBox(NULL, (LPCWSTR)error, L"Ошибка", MB_ICONSTOP);
            exit(1);
        }
        int nRows = m - n + 1;
        CMatrix M(nRows, 1);
        for (int i = n; i <= m; i++)M(i - n, 0) = (*this)(i, k);
        return M;
    }

//-----
CMatrix CMatrix::RedimMatrix(int NewRow, int NewCol)
// Изменяет размер матрицы с уничтожением данных
// NewRow - новое число строк
// NewCol - новое число столбцов
{
    for (int i = 0; i < n_rows; i++) delete array[i];
    delete array;
    n_rows = NewRow;
    n_cols = NewCol;
    array = new double*[n_rows];
    for (int i = 0; i < n_rows; i++) array[i] = new double[n_cols];
    for (int i = 0; i < n_rows; i++)
        for (int j = 0; j < n_cols; j++) array[i][j] = 0;
    return (*this);
}

//-----
CMatrix CMatrix::RedimData(int NewRow, int NewCol)
// Изменяет размер матрицы с сохранением данных, которые можно сохранить
// NewRow - новое число строк
// NewCol - новое число столбцов
{
    CMatrix Temp = (*this);
    this->RedimMatrix(NewRow, NewCol);
    int min_rows = Temp.rows() < (*this).rows() ? Temp.rows() : (*this).rows();
    int min_cols = Temp.cols() < (*this).cols() ? Temp.cols() : (*this).cols();
    for (int i = 0; i < min_rows; i++)
        for (int j = 0; j < min_cols; j++) (*this)(i, j) = Temp(i, j);
    return (*this);
}

//-----
CMatrix CMatrix::RedimMatrix(int NewRow)
// Изменяет размер матрицы с уничтожением данных
// NewRow - новое число строк
// NewCol=1
{

```

```

        for (int i = 0; i < n_rows; i++) delete array[i];
        delete array;
        n_rows = NewRow;
        n_cols = 1;
        array = new double*[n_rows];
        for (int i = 0; i < n_rows; i++) array[i] = new double[n_cols];
        for (int i = 0; i < n_rows; i++)
            for (int j = 0; j < n_cols; j++) array[i][j] = 0;
        return (*this);
    }

//-----
CMatrix CMatrix::RedimData(int NewRow)
// Изменяет размер матрицы с сохранением данных, которые можно сохранить
// NewRow - новое число строк
// NewCol=1
{
    CMatrix Temp = (*this);
    this->RedimMatrix(NewRow);
    int min_rows = Temp.rows() < (*this).rows() ? Temp.rows() : (*this).rows();
    for (int i = 0; i < min_rows; i++) (*this)(i) = Temp(i);
    return (*this);
}

//-----
double CMatrix::MaxElement()
// Максимальное значение элементов матрицы
{
    double max = (*this)(0, 0);
    for (int i = 0; i < (this->rows()); i++)
        for (int j = 0; j < (this->cols()); j++) if ((*this)(i, j) > max) max =
(*this)(i, j);
    return max;
}

//-----
double CMatrix::MinElement()
// Минимальное значение элементов матрицы
{
    double min = (*this)(0, 0);
    for (int i = 0; i < (this->rows()); i++)
        for (int j = 0; j < (this->cols()); j++) if ((*this)(i, j) < min) min =
(*this)(i, j);
    return min;
}

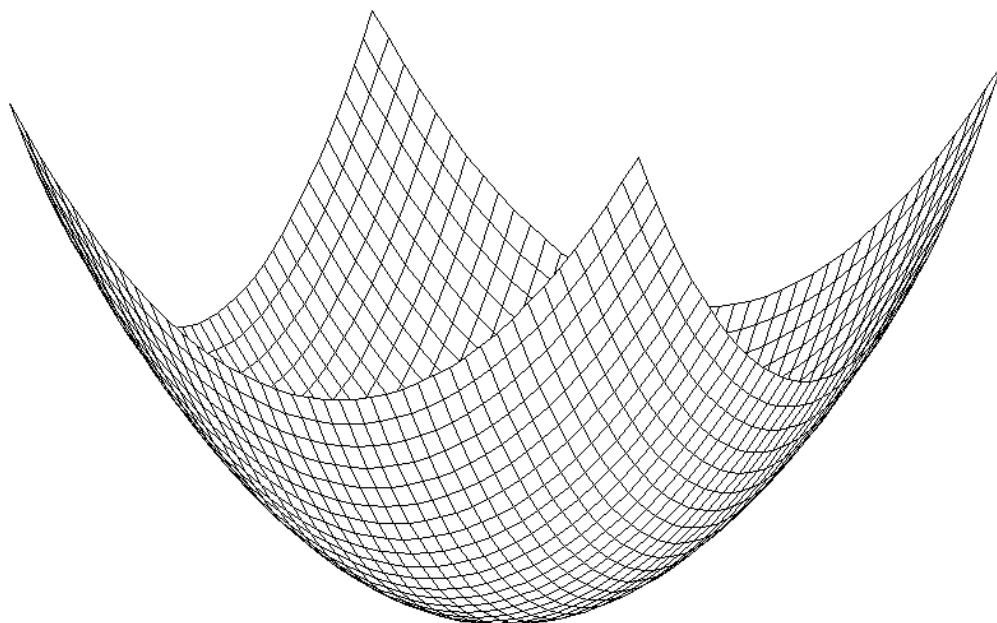
```

## Результат работы:

Lab06

Файл Function

$f=30.0$ ,  $q=45.0$



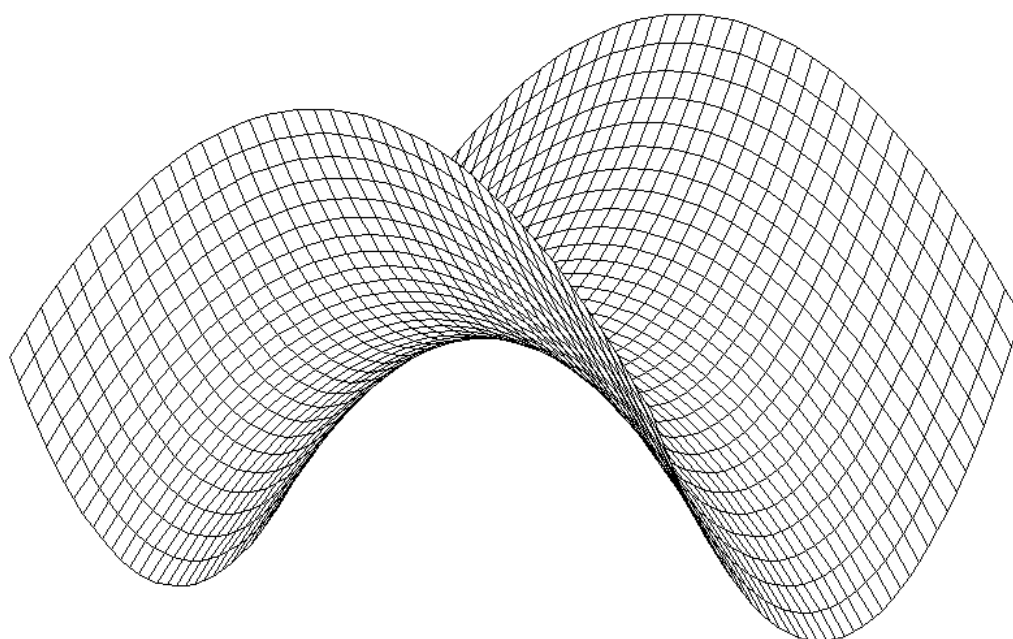
Готово

NUM SCRL

Lab06

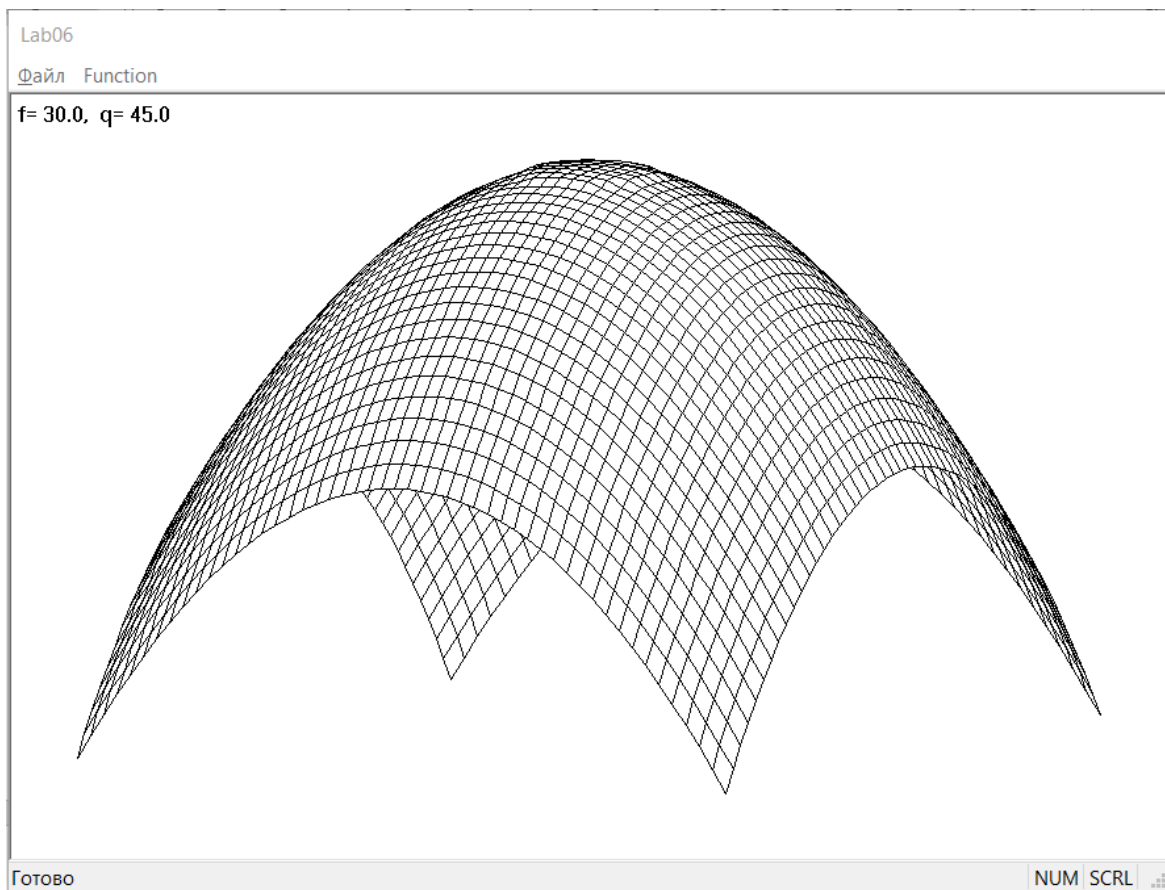
Файл Function

$f=30.0$ ,  $q=45.0$



Готово

NUM SCRL



Литература:

<https://habr.com/ru/post/539420/>

<https://helpiks.org/7-85986.html>